Exercise 1a

Loop heavy R function to calculate Euclidean distance between two coordinate matrices.

```
r.iDist <- function(coords.1, coords.2){

    n.1 <- nrow(coords.1)
    n.2 <- nrow(coords.2)
    p <- ncol(coords.1)

    D <- matrix(0, n.1, n.2)

    dist <- 0

    start.time <- proc.time()

    for(i in 1:n.1){
        for(j in 1:n.2){
            dist <- 0
            for(k in 1:p){
                dist <- dist + (coords.1[i,k] - coords.2[j,k])^2
            }
            D[i,j] <- sqrt(dist)
        }
    }

    return(list("D"=D, sys.time=proc.time()-start.time))
}
```

Similar Euclidean distance function in C/C++ via `.Call()`.

`R` file

```
c.iDist <- function(coords.1, coords.2){

    if(!is.matrix(coords.1))
        coords.1 <- as.matrix(coords.1)

    if(missing(coords.2))
        coords.2 <- coords.1

    if(!is.matrix(coords.2))
        coords.2 <- as.matrix(coords.2)

    if(ncol(coords.1) != ncol(coords.2))
        stop("error: ncol(coords.1) != ncol(coords.2)")

    p <- ncol(coords.1)
    n1 <- nrow(coords.1)
    n2 <- nrow(coords.2)

    D <- matrix(0, n1, n2)

    storage.mode(coords.1) <- "double"
    storage.mode(coords.2) <- "double"
    storage.mode(D) <- "double"
    storage.mode(n1) <- "integer"
    storage.mode(n2) <- "integer"
    storage.mode(p) <- "integer"

    start.time <- proc.time()

    .Call("cIDist", coords.1, n1, coords.2, n2, p, D)

    list("D"=D, sys.time=proc.time()-start.time)
}
```

C++ file

```
#include <R.h>
#include <Rmath.h>
#include <Rinternals.h>

extern "C" {

  SEXP cIDist(SEXP coords1_r, SEXP n1_r, SEXP coords2_r, SEXP n2_r, SEXP p_r, SEXP D_r){

    double *coords1 = REAL(coords1_r);
    int n1 = INTEGER(n1_r)[0]; // R treat scalar as pointer.

    double *coords2 = REAL(coords2_r);
    int n2 = INTEGER(n2_r)[0];

    int p = INTEGER(p_r)[0];

    double *D = REAL(D_r);

    int i, j, k;
    double dist = 0.0;

    for(i = 0; i < n1; i++){
      for(j = 0; j < n2; j++){
          dist = 0.0;
            for(k = 0; k < p; k++){
              dist += pow(coords1[k*n1+i]-coords2[k*n2+j],2);
            }
          D[n1*j+i] = sqrt(dist);
      }
    }

    return(R_NilValue);
  }
}
```

Add some parallelization to the distance function.

`R` file

```
c.iDist.omp <- function(coords.1, coords.2, n.omp.threads=1){

    if(!is.matrix(coords.1))
        coords.1 <- as.matrix(coords.1)

    if(missing(coords.2))
        coords.2 <- coords.1

    if(!is.matrix(coords.2))
        coords.2 <- as.matrix(coords.2)

    if(ncol(coords.1) != ncol(coords.2))
        stop("error: ncol(coords.1) != ncol(coords.2)")

    p <- ncol(coords.1)
    n1 <- nrow(coords.1)
    n2 <- nrow(coords.2)

    D <- matrix(0, n1, n2)

    storage.mode(coords.1) <- "double"
    storage.mode(coords.2) <- "double"
    storage.mode(D) <- "double"
    storage.mode(n1) <- "integer"
    storage.mode(n2) <- "integer"
    storage.mode(p) <- "integer"
    storage.mode(n.omp.threads) <- "integer"

    start.time <- proc.time()

    .Call("cIDistOMP", coords.1, n1, coords.2, n2, p, D, n.omp.threads)


    list("D"=D, sys.time=proc.time()-start.time)
  }
```

C++ file

```
#include <R.h>
#include <Rmath.h>
#include <Rinternals.h>

#ifdef _OPENMP
#include <omp.h>
#endif

extern "C" {

  SEXP cIDistOMP(SEXP coords1_r, SEXP n1_r, SEXP coords2_r, SEXP n2_r, SEXP p_r, SEXP D_
r, SEXP nThreads_r){

    double *coords1 = REAL(coords1_r);
    int n1 = INTEGER(n1_r)[0];

    double *coords2 = REAL(coords2_r);
    int n2 = INTEGER(n2_r)[0];

    int p = INTEGER(p_r)[0];

    double *D = REAL(D_r);

    int nThreads = INTEGER(nThreads_r)[0];

#ifdef _OPENMP
    omp_set_num_threads(nThreads);
#else
    if(nThreads > 1){
      warning("n.omp.threads = %i requested however source code was not compiled with Op
enMP support.", nThreads);
      nThreads = 1;
    }
#endif

    int i, j, k;

    double dist = 0.0;

#ifdef _OPENMP
#pragma omp parallel for private(j, dist, k)
#endif
    for(i = 0; i < n1; i++){
      for(j = 0; j < n2; j++){
          dist = 0.0;
            for(k = 0; k < p; k++){
              dist += pow(coords1[k*n1+i]-coords2[k*n2+j],2);
            }
          D[n1*j+i] = sqrt(dist);
      }
    }

    return(R_NilValue);
```

```
  }
}
```

Compile the C++ shared objects.

```
system("R CMD SHLIB cIDist.cpp")

system("R CMD SHLIB cIDistOMP.cpp")
```

Run some time tests using the three `IDist` functions.

```
##Load shared libraries
source("rIDist.R")

dyn.load("cIDist.so")
source("cIDist.R")

dyn.load("cIDistOMP.so")
source("cIDistOMP.R")

##Make data
n.1 <- 5000
coords.1 <- cbind(1:n.1, 1:n.1)

n.2 <- 5000
coords.2 <- cbind(1:n.2, 1:n.2)

##Calculate Euclidean distance matrices and print timing
r.D <- r.iDist(coords.1, coords.2)
print(r.D$sys.time)
```

```
##    user  system elapsed
##  15.198   0.149  15.497
```

```
c.D <- c.iDist(coords.1, coords.2)
print(c.D$sys.time)
```

```
##    user  system elapsed
##   0.193   0.002   0.200
```

```
c.omp.D <- c.iDist.omp(coords.1, coords.2, n.omp.threads=2)
print(c.omp.D$sys.time)
```

```
##    user  system elapsed
##   0.269   0.005   0.148
```